On Neighborhood Tree Search

Houda Derbel
FSEGS
Route de l'aéroport km 4, Sfax
Tunisia
derbelhouda@yahoo.fr

Bilel Derbel
Université Lille 1
LIFL CNRS – INRIA Lille
France
bilel.derbel@lifl.fr

ABSTRACT

We consider the neighborhood tree induced by alternating the use of different neighborhood structures within a local search descent. We investigate the issue of designing a search strategy operating at the neighborhood tree level by exploring different paths of the tree in a heuristic way. We show that allowing the search to 'backtrack' to a previously visited solution and resuming the iterative variable neighborhood descent by 'pruning' the already explored neighborhood branches leads to the design of effective and efficient search heuristics. We describe this idea by discussing its basic design components within a generic algorithmic scheme and we propose some simple and intuitive strategies to guide the search when traversing the neighborhood tree. We conduct a thorough experimental analysis of this approach by considering two different problem domains, namely, the Total Weighted Tardiness Problem (SMTWTP), and the more sophisticated Location Routing Problem (LRP). We show that independently of the considered domain, the approach is highly competitive. In particular, we show that using different branching and backtracking strategies when exploring the neighborhood tree allows us to achieve different trade-offs in terms of solution quality and computing cost.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving and Search—*Heuristic methods*

General Terms

Algorithms,

Keywords

Metaheuristics, neighborhood combination, VND, VNS.

1. INTRODUCTION

Context and Motivation: Metaheuristics are now considered as a well established algorithmic framework providing flexible and powerful tools to solve many hard optimization problems. Many efforts are being made by the research community in order to develop new search methods to help the design of both effective and efficient algorithms. In this paper, we build on previous techniques by developing an intuitive idea based on exploiting different neighborhoods in a forward-backward manner to explore what we term *the neighborhood tree*. Generally speaking, we consider the possibility of making backward moves to a solution previously explored by some neighborhoods, and continue the search from there using other different neighborhoods searching for a better neighborhood combination. In the following, we first review some previous related works, then after, we give our contribution and describe our findings in more details.

Background and related works: Among other search techniques, variable neighborhood search (VNS) and its several variants [14] are based on the systemic change of neighborhood within the search. For instance, Variable Neighborhood Descent (VND) exploits the idea of alternating between several neighborhoods within an iterative local improvement descent to escape local optima. More precisely, starting with a first neighborhood structure, VND performs local search until no further improvements are possible. From this local optimum, the local search is continued with the next neighborhood. If an improving solution is found, then the local search continues with the first neighborhood, otherwise the next available neighborhood is explored, and so on until no further improvements can be obtained. It is well known that the performance of VND can highly depend on the order the neighborhoods are alternated. In standard variants of VND, it is often admitted that ordering neighborhoods in an increasing cost/size is a reasonable strategy. However, this standard strategy is not always applicable, for instance, when the best ordering for a given problem can vary from one instance to another one. Actually, the issue of how to combine/exploit/search different neighborhoods is not new and one can find many different studies on the subject. For instance, in [22], a fast relaxation of neighborhoods is evaluated in order to select the most accurate ones. In [16], a self-adaptive strategy is used to rank neighborhoods and to dynamically choose the best suited ordering. A number of specific multi-neighborhood combination functions can also be found. For instance, many studies consider to take the union of some basic neighborhoods. The so-called neighborhood composition and the token-ring search are also other well known neighborhood combination functions, see e.g., [18, 10, 12, 17]. More generally, hyperheuristics [4] can be considered as a high level approach operating in the neighborhood space and aiming at producing effective hyper-search strategies. For instance, in [6, 21], simple hyperheuristic selection strategies are considered where low level heuristics (neighborhoods) are chosen either randomly, or greedily, or based on a score function. Several other sophisticated hyper-strategies, mainly inspired by the way metaheuristics operate, can be found in the literature, see e.g., [5].

GECCO'12, July 7-11, 2012, Philadelphia, Pennsylvania, USA.

Technique overview and results: The study conducted in this paper is based on the simple observation that defining how neighborhoods are alternated within a local search descent is nothing other than defining a specific strategy to traverse a neighborhood tree, where the root of the tree represents the initial candidate solution and intermediate nodes represent solutions obtained by applying one of the possible neighborhoods. In other words, we view the trajectory of a variable neighborhood search as a high level neighborhood path, where path nodes are solutions and every path hop represents the exploration of one solution using one neighborhood among those available. Following this observation, we term *a neighborhood tree search (NTS)* a strategy which is able to traverse the neighborhood tree efficiently searching for promising paths. It should be clear that a systematic traversal (exploration of all neighborhood branches) could not be efficient especially when the number of neighborhoods is high.

In this paper, we focus on the possibility of backtracking to previously visited solutions while branching and pruning tree nodes all along a search path. We show that this idea with basic iterative improvement descents leads to efficient search strategies both in terms of solution quality and computing cost. More specifically, we consider a simple randomized neighborhood selection strategy, where the choice of which neighborhood to select at runtime is made uniformly at random among those not yet explored. When effectively branching a neighborhood, we consider both deterministic and randomized adaptive strategies, basically relying on the neighborhood path traversed by the search in previous rounds. As for backtracking, we investigate intuitive strategies based on random and tournament selection techniques. We would like to emphasize that the proposed approach and its design components are *generic* and *not* specific to a fixed problem *nor* to any particular neighborhood class.

We study the properties of the proposed approach by considering several instances coming from two different and well-studied problem domains: the Single Machine Total Weighted Tardiness Problem (SMTWTP) in the family of scheduling problems, and Location Routing Problem (LRP). Both problems are NP-Hard. Many previous studies have been successfully applied to solve the SMTWTP using hybrid variable neighborhood like searches. LRP is a more sophisticated problem which involves two simultaneous decisions: which depots to open and what routes to plan. Common to these two problems, many natural neighborhood structures can be considered making them two excellent case studies to analyze how our neighborhood tree based approach performs under different scenarios. Through extensive experiments, we show that our approach leads to substantial improvements in the solving of the two considered problems. More specifically, for SMTWTP we consider three neighborhood structures and we show that NTS performs better than standard VND executed with any neighborhood ordering, i.e., NTS is able to dynamically find its way along the neighborhood tree without any specific tuning. For LRP, we consider eleven neighborhoods and a finely tuned VNS algorithm. Ultimately, we show that NTS is able to beat VNS without requiring any specific perturbation/shaking phase, but the backtracking it-slef. More importantly, VNS is used as a base-line algorithm allowing us to show how NTS performs when instantiating its components following different strategies. This allows us to give insights into the behavior of NTS and to better understand its critical design issues. In particular, we show that NTS can lead to different (and incomparable) trade-offs in terms of solution quality and running time. In a general point of view, our study reveals that NTS is a promising approach offering many interesting search abilities.

Outline: In Section 2, we give an algorithmic scheme for NTS and describe intuitive strategies to be analyzed later. In Section 3, we describe the considered problems and neighborhoods. In Section 4 (resp. 5), we analyze NTS and give our experimental results.

2. NEIGHBORHOOD TREE SEARCH (NTS)

2.1 Preliminaries

Let us assume that we are given an optimization problem and a set of corresponding neighborhoods. We aim at designing an algorithm that exploits those neighborhoods as efficiently as possible. For simplicity, assume in addition that we have a *step function* that given a candidate solution s returns a solution s' computed w.r.t. one neighborhood. Having an initial solution s_0, we then term the neighborhood tree $\mathcal{T}$ the (possibly infinite) tree structure obtained by the following process. The root node of $\mathcal{T}$ is the initial solution s_0. The first level of $\mathcal{T}$, are the candidate solutions obtained from s_0 by applying the previously defined *step function* w.r.t. every available neighborhood. The j^{th} level is then constructed recursively from the internal nodes in the $(j-1)^{\text{th}}$ level and so on. Notice that this is a rather informal definition which is only given for the sake of illustration and to clarify our preliminary remarks.

It is clear that designing a local search algorithm exploiting the available neighborhoods can be viewed as designing a specific strategy to explore the considered neighborhood tree. This is what we term a neighborhood tree search (NTS) algorithm, i.e., a traversal strategy of the neighborhood tree searching for paths leading to promising regions. Designing such a traversal search algorithm can be difficult for many reasons. Firstly, for many optimization problems there may exist a relatively high number of natural neighborhood structures, say a constant $k > 2$. Hence, a trivial exhaustive traversal of all tree nodes at height $g(n)$, a function of the problem size n, would require at least order of $k^{g(n)}$ steps, which can be intractable. Secondly, the goal is not to systematically traverse as many as possible tree nodes, but to find a path leading to high quality solutions while paying the minimum computing cost. Fortunately, we know that there exist heuristic traversal techniques leading to relatively efficient and effective search algorithms. For instance, this is the case for VND like search algorithms and many others that could be viewed as specific traversal strategies. In this paper, we focus on designing dynamic traversal heuristics where at each step, one have to decide what neighborhood to consider when going deeper in the neighborhood tree while backtracking to a previously visited solution whenever the search stacks into non promising tree regions.

2.2 A basic randomized NTS

Algorithm 1 gives a relatively detailed description of our first NTS example. As input, we assume that we are given a set of k neighborhood structures $\{\mathcal{N}_1, \cdots, \mathcal{N}_k\}$ relative to a given problem and a fitness/evaluation function f to be minimized. Algorithm 1 maintains a trajectory path (variable $Path$) containing an ordered sequence of visited solutions with their neighborhood usage. This is encoded by variable $h_s = (h_s^1, \cdots, h_s^k)$ where h_s^i is 1 whenever neighborhood $\mathcal{N}_i$ has been used to explore solution s and 0 otherwise. The algorithm then proceeds iteratively by considering the last solution s (function HEAD) appearing in the trajectory path. For that solution, a neighborhood $\mathcal{N}_i$ is chosen uniformly at random among those that have not been used to explore s. A STEP function is then applied to compute a new solution s' and neighborhood usage variable h_s^i is updated. The STEP function could be for

instance any local search heuristic based on neighborhood structure $\mathcal{N}_i$, e.g., a hill-climbing. Having explored a new neighborhood, we shall decide whether to continue the search with solution s' or to backtrack. In Algorithm 1, a simple acceptance criterion is used. More precisely, if the new explored solution s' is found to improve s then, we make a move forward by pushing s' at the end the trajectory path. Otherwise, we check whether there exists some neighborhoods which have *not* been used to explore s. If such a situation exists, we simply continue the inner-loop, that is we try to find an improving solution by selecting uniformly at random a non explored neighborhood w.r.t. the current solution s. Otherwise, a backtrack move is activated ('else' condition). Notice that in this case, current solution s is not necessarily a local optimum w.r.t. all neighborhoods. In fact, this depends on the STEP function and the depth of s in the search trajectory. Backtracking in Algorithm 1 is done using a simple uniform randomized selection process. More precisely, among path solutions which are not yet explored by all neighborhoods, one is chosen uniformly at random, say solution s_j at position j. The trajectory path is then updated by deleting those solutions laying between s_j and s. The search then continues from s_j in the same way until the trajectory path becomes empty.

Algorithm 1: A simple randomized variant of NTS

Input: A set of k neighborhood structures $\{\mathcal{N}_1, \cdots, \mathcal{N}_k\}$
$s \leftarrow$ initial solution ;
$(h_s^1, \cdots, h_s^k) \leftarrow (0, \cdots, 0)$; $Path \leftarrow \{(s, (h_s^1, \cdots, h_s^k))\}$;
repeat
  /** Current trajectory solution **/
  $(s, h_s) \leftarrow$ HEAD$(Path)$;
  /** Neighborhood Selection **/
  $I_s \leftarrow \{\ell \mid h_s^\ell = 0\}$;
  $i \leftarrow$ RANDOM(I_s) ;
  /** Neighborhood Exploration **/
  $s' \leftarrow$ STEP$(s, \mathcal{N}_i)$;
  $h_s^i \leftarrow 1$;
  /** Move or Backtrack **/
  if $f(s) < f(s')$ **then**
    $h_{s'} \leftarrow (0, \cdots, 0)$;
    $Path \leftarrow$ PUSH$\big((s', h_{s'}), Path\big)$;
  else if $|I_s| + 1 = k$ **then**
    $Path \leftarrow$ RANDOM_BACKTRACK$(Path)$;
until $Path = \emptyset$;

2.3 NTS generic scheme and variants

Algorithm 1 given in the previous section is clearly a specific implementation of the more generic scheme given in Algorithm 2. Generally speaking, Algorithm 2 is in fact an attempt to give a high level procedural description of a NTS like algorithm. In particular, we identify the *history* variable which is typically used to record information about search trajectory and neighborhood performance. We also have the neighborhood selection stage which role is to help the search going towards promising neighborhood branches. The STEP and ACCEPT function, play the role of branching/pruning. These two functions should be thought in the same way classical local search algorithms operate, but keeping in mind that possibly several neighborhoods can be used. The third main stage of Algorithm 2 is backtracking. Within NTS, backtracking serves mainly to adapt the traversal when it is stack into paths that do not lead to improvements. Backtracking should be thought with respect to search history. In this paper, we study the properties of NTS by considering the following particular variants.

Algorithm 2: general purpose design scheme for NTS

Input: A set of k neighborhood structures $\{\mathcal{N}_1, \cdots, \mathcal{N}_k\}$.
$s \leftarrow$ initial solution; $history \leftarrow \varnothing$;
repeat
  /** Neighborhood Selection Strategy **/
  $i \leftarrow$ SELECT$(s, history)$;
  /** Branching/Pruning Strategy **/
  $s' \leftarrow$ STEP$(s, \mathcal{N}_i, history)$;
  if ACCEPT$(s, s', history)$ **then**
    $s \leftarrow s'$;
  else
    /** Backtracking Strategy **/
    $s \leftarrow$ BACKTRACK$(history)$;
until STOPPING_CONDITION ;

Trajectory history: In all our variants, we record the path traversed by the search and containing the branching solutions and their relative neighborhood usage (exactly in the same way than in Algorithm 1). We additionally record the (local) best fitness f_s^{best} relative to each solution s and observed by the search when the local step function STEP$(s, \mathcal{N}_i)$ is applied at position s.

Neighborhood Selection Strategy: We simply consider the randomized process depicted in Algorithm 1, i.e., one neighborhood among those not previously used at current path position is selected uniformly at random.

Step function STEP$(s, \mathcal{N}_i)$: we study four classical alternatives denoted BI (best improvement), FI (first impr.), BD (best descent) and FD (first descent). BI corresponds to the case where solution s' is the neighbor of s (w.r.t $\mathcal{N}_i$) with the best fitness. For strategy FI, s' is the first neighbor of s which is found to have a better fitness than s, when processing s neighbors in a random order. BD (resp. FD) denotes a local search descent where BI (resp. FI) strategy is applied until no improving neighbors can be found.

Acceptance/Branching criterion: We study three strategies denoted AA, AI, and AT. AA is exactly the same than in Algorithm 1, i.e., any solution s' that improves the fitness of current solution s is accepted : $f(s') < f(s)$. AI denotes the strategy where solution s' is accepted if its fitness s is better than f_s^{best}, i.e., $f(s') < f_s^{\text{best}}$. AT is a combination of AA and AI. More specifically, a solution s' is always accepted if $f(s') < f_s^{\text{best}}$. Otherwise, if $f(s') > f_s^{\text{best}}$, but $f(s') < f(s)$ then s' is accepted with a probability parameter p_a. Otherwise s' is not accepted. In our study, we adopt an adaptive strategy where $p_a = 1/d(s)$ with $d(s)$ the position of solution s in the trajectory path. In other words, a neighborhood, leading to a branch improving s, but not improving the previous best local fitness obtained using a different neighborhood, is accepted with a probability which is proportional to the branch height in the neighborhood tree, i.e., the more we are deep in the neighborhood tree, the more it is unlikely to explore non improving branches.

Backtracking strategy: We consider three backtracking strategies denoted BR, BH, BU. BR is the strategy depicted in Algorithm 1, i.e., among path positions which are not yet explored by all neighborhoods, one is chosen uniformly at random. BH and BR are more sophisticated tournament-based selection strategies. More precisely, for both BH and BR, we select two distinct path positions s_j and $s_{j'}$ uniformly at random among those not yet explored by all neighborhoods. With BH, we backtrack to the solution which

is less deep in the trajectory path, i.e., if $s_{j'}$ appears after s_j in the search path, then we backtrack to s_j. With BU, we backtrack to the solution which was explored less often by available neighborhoods.

Stopping Condition: We consider two different stopping conditions: we end the search when (i) the path trajectory is empty, or (ii) a maximum number of fitness evaluations is reached.

Terminology and notations: For clarity, we shall use the following notation **NTS-**$(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ where $\mathbf{X} \in \{\mathbf{FI}, \mathbf{BI}, \mathbf{FD}, \mathbf{BD}\}$ is the step function, $\mathbf{Y} \in \{\mathbf{AA}, \mathbf{AI}, \mathbf{AT}\}$ the branching/acceptance strategy, and $\mathbf{Z} \in \{\mathbf{BR}, \mathbf{BH}, \mathbf{BU}\}$ is the backtracking strategy.

3. PROBLEM DOMAINS

3.1 Single machine scheduling (SMTWTP)

Problem definition and motivation: In the Single Machine Total Weighted Tardiness Problem, we are given n jobs. Each job has to be processed without any interruption on a single machine that can only process one job at a time. Each job has a processing time p_j , a due date d_j and an associated weight w_j (reflecting the importance of the job). The tardiness of a job j is defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of job j in the current sequence of jobs. The goal is then to find a job sequence minimizing the sum of the so-called weighted tardiness: $\sum_{i=1}^{n} w_i \cdot T_i$. SMTWTP is NP-hard. Several different metaheuristics have been proved to efficiently solve SMTWTP benchmark instances, e.g., [11, 3, 13] to cite a few. SMTWTP is in fact a well understood problem which is often used to study the properties of search heuristic methods. This paper is not an exception. Although we are able to show that very simple NTS techniques outperform previous more sophisticated and finely tuned heuristics for SMTWTP, we shall rather focus on studying and understanding the behavior of our NTS heuristics.

Neighborhoods: Permutations are the standard representation used for SMTWTP. In this paper, we consider three standard neighborhoods. *Exchange (E)*: all permutations that can be obtained by swapping adjacent jobs in the permutation. *Swap (S)*: all permutations that can be obtained by swapping adjacent jobs at the i^{th} and j^{th} position. *Insert (I)*: all permutations that can be obtained by removing a job at position i and inserting it at position j.

Instances: We consider the well known 100 Job instance set, and at a less extent, the 50 and 40 Job instances from the OR-Library [1]. Each instance set contains 125 instances.

3.2 Location Routing problem (LRP)

Problem definition and motivation: LRP [20, 19] deals with two NP-hard problems, namely, facility location problem (FLP) and vehicle routing problem (VRP). Roughly speaking, in LRP one has to simultaneously decide which depots to open and what routes to establish to satisfy client demands. Besides being a challenging problem for local search heuristics, LRP is of special interest since many neighborhoods can be naturally considered for both the location and the routing level (which are known to be inter-dependent). Many specific search strategies have been studied for LRP, e.g., [8, 7, 9] to cite a few. A common aspect in these studies is to find a good balance for simultaneously searching the routing level and the location level. In particular, there exist a rich literature on several different neighborhoods dealing with the two LRP levels. This makes the choice and the combination of neighborhoods critical and thus LRP is an excellent candidate problem to study our approach. In this paper, we consider the uncapacitated vehicles and capacitated LRP [2]. More specifically, we consider a set of n customers and a set of m potential depots. Each depot has a limited capacity and a fixed opening cost. Each depot is associated with a single uncapacitated vehicle. Each customer has a non-negative demand which is known in advance and should be satisfied. For any pair of clients (or client-depot), there is an associated traveling cost. LRP then consists in minimizing the total cumulative cost of both depot opening (location) and client delivery (Routing).

Neighborhoods: We consider a natural representation of a candidate solution (not necessarily feasible) for LRP, namely, a list of opened and non opened depots. For each depot, a permutation represents the assigned clients and their order in the route. We consider eleven neighborhoods sketched in the following. $\mathcal{N}_1$ (resp. $\mathcal{N}_2$): All solutions obtained by performing a client *insertion* move in the permutation(s) encoding one single depot route (resp. two *different* depot routes). $\mathcal{N}_3$ (resp. $\mathcal{N}_4$): All solutions obtained by performing a swap move on the permutation(s) encoding one depot route (resp. two depot routes). $\mathcal{N}_5$ (resp. $\mathcal{N}_6$): All solutions obtained by performing a classical 2-opt move on the permutation(s) encoding one depot route (resp. two depot routes). $\mathcal{N}_7$ (resp. $\mathcal{N}_8$): All solutions obtained by performing an extended insertion move on the permutation(s) encoding every route (resp. two *different* routes), that is an insertion of any sub-route of any possible length, i.e., route bone insertion. $\mathcal{N}_9$ (resp. $\mathcal{N}_{10}$): All solutions obtained by performing a route bone insertion move as for neighborhoods $\mathcal{N}_7$ (resp. $\mathcal{N}_8$) but while inserting clients sub-route in the reverse order. $\mathcal{N}_{11}$: All candidate solutions obtained by closing a depot and affecting its whole route to a closed depot (close one depot and open a new one).

Since we are dealing with neighborhoods producing possibly unfeasible solutions, we use an evaluation function f defined as following: $f(s) = c(s) + p(s)$ where $c(s)$ is the cost of s as stated by the objective function of LRP and $p(s)$ is a penality on the violation of depot capacity constraints. It is calculated by the equation: $p(x) = \sum_j \alpha \cdot \max\{0, Q_j(s) - b_j\}$ where α is a weight factor parameter, $Q_j(s)$ is the total demand of customers serviced by depot j and b_j is the capacity associated with depot j, i.e., the more depot constraints are violated, the more is the penality and the more the search is forced to move toward feasible regions.

Instances: We consider a set of 450 instances taken from the literature [2]. These instances can be grouped into finely defined classes according LRP specific parameters. In our experimental results, we simply group them into 5 sets according to the number of clients (n) and the number of depots (m): $(n, m) \in \{(5, 10), (5, 20), (5, 30), (10, 20), (10, 30)\}$. Each set is containing equally the same number of instances, i.e., 90.

4. EXPERIMENTAL ANALYSIS: SMTWTP

4.1 Results with standard VND

For SMTWTP, we consider to study the behavior of our approach compared to standard VND techniques. In Table 1, we report a summary of results we have obtained for SMTWTP when running a standard VND using the 6 possible ordering of neighborhoods and the 4 possible strategies for making a local step move. In accordance with results reported in previous studies, e.g., [11], VND is well suited for solving SMTWTP. In fact, the average percentage deviation from optimal is relatively low and around 90% of instances are solved to optimality by at least one trial over 30 performed in our experiments. However, one can notice that some instances remain hard to solve by standard VND as reported in previous works. In addition, no fixed ordering nor local step strategy outperforms all the others for all three measures reported in Table 1.

4.2 Results overview with NTS

Table 1: Results for standard VNDs with a random initial solution. Results are for the 100 job instances with 30 trials per instance. First column gives neighborhood ordering, i.e., Exchange (E), Insert (I), Swap (S). FI, BI, FD and BD columns are for the local step functions given in Section 2. N_{opt} is the number of optimal solutions found by at least one trial. $\overline{\Delta}$ is the average percentage deviation from optimal and $\overline{\text{Eval}}$ is the average number of evaluations until search termination. Bold style is for best result (for each column).

Order	FI			BI			FD			BD		
	N_{opt}	$\overline{\text{Eval}}$	$\overline{\Delta}$	N_{opt}	$\overline{\text{Eval}}$	$\overline{\Delta}$	N_{opt}	$\overline{\text{Eval}}$	$\overline{\Delta}$	N_{opt}	$\overline{\text{Eval}}$	$\overline{\Delta}$
EIS	102	140872.4	**0.010**	**103**	**471156.4**	**0.015**	**113**	121787.3	0.015	101	**667976.4**	0.019
ESI	**106**	121372.3	0.016	91	485211.7	0.018	102	106214.4	0.015	92	946366.1	0.019
IES	104	**120262.4**	0.017	96	1024409.3	0.022	95	101053.6	0.017	93	1006583.9	0.021
ISE	101	135572.1	0.013	96	865525.5	0.022	112	115144.1	**0.014**	103	871254.1	**0.018**
SEI	98	121220.9	0.017	93	1026960.0	0.021	102	**100903.9**	0.016	92	1007357.2	0.025
SIE	**106**	138007.4	0.015	101	864900.3	0.023	106	114412.0	**0.014**	**105**	871631.1	0.021

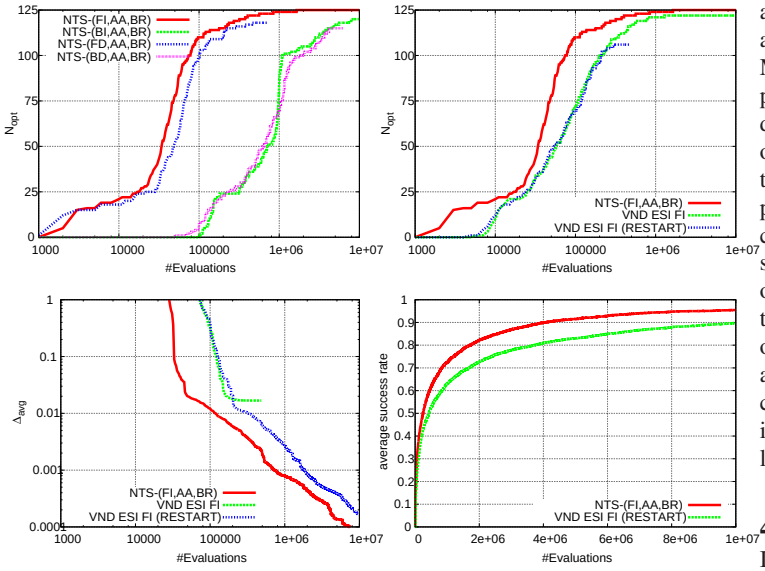

Figure 1: Results for NTS-(*,AA,BR) Vs standard VND for 100 job instances (labels refer to search strategies). Top: cumulative number of instances solved to optimality (N_{opt}) by at least one trial over 30 as a function of number of fitness evaluations. Bottom-Left: Evolution, with number of evaluations, of the average percentage deviation from optimal ($\overline{\Delta}$) averaged over 30 trials. Bottom-Right: Evolution, with number of evaluations, of the success rate averaged over the 125 instances.

For SMTWTP, the reported results are obtained with the simple variant of NTS given in Algorithm 1, with random initial solution, acceptance strategy AA, and backtracking strategy BR, i.e., NTS-$(*, AA, BR)$. As stopping condition, the search terminates when either a maximum number of evaluations, namely 10^7, is reached, or the trajectory path becomes empty. Our results for NTS are summarized in Fig. 1. Firstly, we remark that for both FD and BD local step strategies, the search terminates before the maximum number of evaluations is reached (Fig. 1 Top-left). At the opposite, for both FI and BI the search continues without the backtracking being able to force termination. This is mainly due to the relatively high trajectory path length as we will discuss later. Furthermore, as could be expected, first improvement strategies are less costly compared to best improvement strategies. We also found that FI outperforms all other step strategies both in computing cost and number of instances solved to optimality. In fact, it is the only strategy which is able to find an optimal solution (over the 30 performed trials) for all the 125 instances.

Moreover, NTS using step functions FI and FD can be proved to provide substantial improvements in all aspects over all the standard VND variants reported in Table 1. For the sake of clarity, we only report our finding using one VND ordering, namely ESI. Notice however that similar conclusions can be drawn for the other possible orderings. To be fair in our comparative study, we further consider restarting the VND algorithm from a randomly generated solution in the case VND terminates before the maximum number of evaluations is reached. As shown in Fig.1 (Top-right and Bottom), NTS outperforms VND both in terms of: cumulative number of instances solved to optimality (N_{opt}), average percentage deviation from optimal ($\overline{\Delta}$), and average success rate that is the percentage of trials that do find the optimal solution, i.e., this can be interpreted as the probability distribution of finding the optimal solutions for all instances.

4.3 Run Time Distribution Analysis

In previous section, we showed that NTS performs better than VND in general, i.e., results are mainly averaged over instances. In this section, we go to a throughout comparative study. More specifically, we analyze the run-time behavior of NTS compared to standard VND by using run-time distributions (RTD) [15]. RTDs give the cumulative empirically observed probability of finding an optimal solution (or a solution within a specific quality bound) for a given instance as a function of the CPU time. In our study, we use a slightly different definition, where probability is considered as a function of number of evaluations. This is mainly to stay independent of any specific implementation or operating system issues (NTS running time issues are however studied in next section). We examined the behavior of NTS with different step functions and for several instances, mainly, those who are reputed to be relatively hard. In Fig. 2 we present our results for only four instances, but similar conclusions can be made for the others. The RTDs clearly shows that NTS performs better than VND for three out of the four instances, namely, 19, 38, and 41, even if optimality is not required. For instance 86, the difference is less pronounced, with a small advantage in favor of NTS for fitness deviation of 1% from optimal.

4.4 NTS history analysis

Here, we give some basic observations about trajectory path used by NTS. In Table 2, we report the maximum length h_{max} of the trajectory path ever observed for any instance and any trial , and $\overline{h_{max}}$ the maximum length (30 trials per one instance) averaged over 125 instances of each problem size set.

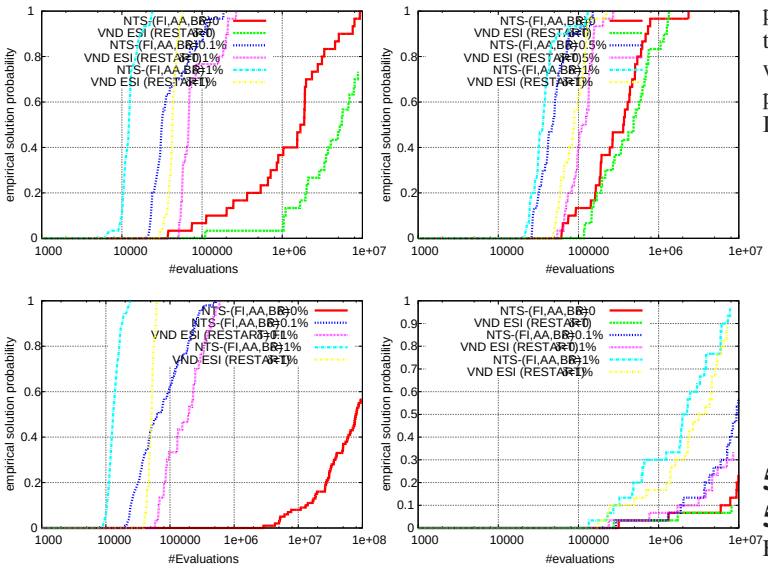

Figure 2: RTDs for NTS *vs* VND with random restart. The x-axis gives the logarithm of the number of fitness evaluations, the y-axis the cumulative empirical solution quality probability. δ denotes the gap (in percentage) between the required solution quality and the optimal solution. Top-Left figure (resp. Top-Right, Bottom-Left and Bottom-Right) shows the results for instance 19 (resp. 38, 42, and 86).

With the FD step function, the maximum length for the three instance sets is at most 8 which is relatively very low. This is a crucial observation that can be exploited in different manners for backtracking. Let h be the maximum length that can be observed which is also the maximum height for the neighborhood tree. Assuming that the exploration of each of the k neighborhoods by the step function FD requires a polynomial time in the problem size, say $p(n)$, then an exhaustive neighborhood tree traversal would lead to an NTS algorithm with computing complexity of roughly $O(k^h \cdot p(n))$. If h is proved to be a constant or even a very small function in n, then an exhaustive neighborhood tree traversal could lead to a relatively efficient NTS. In our experiments, h seems to be very insensitive to n which suggests that other backtracking strategies, e.g., using a fixed/adaptive number of backtrack steps, could improve the search.

For the FI step function, it is clear that (compared to FD) there is a significant increase in the trajectory path length. Therefore, the previous discussion does not hold anymore using FI. In fact, assuming that $h_{\max} = \Omega(n)$, which seems to be the case, an exhaustive search of the neighborhood tree is normally intractable. This is further confirmed by the fact that in all our experiments with the FI strategy, NTS always reaches the maximum number of iterations without emptying the whole search path. We however remark that using random backtracking (BR), FI produces better results than FD. We attribute this to the fact that FI implies a neighborhood tree of relatively high size but relatively diversified neighborhood paths.

To conclude this section, we would like to give some remarks on the impact of maintaing and accessing the history path on CPU running time. As discussed before, path trajectory is relatively low compared to problem size. Knowing that the size of many neighborhood structures is at least linear, and many often polynomial, in problem size, the cost of maintaining the path history stays relatively marginal. In our implementation using standard Java library, without any specific code optimization, the average CPU-time to perform one evaluation for problem size 100 on a standard 2.0 Ghz Intel processor is 0.0005 millisecond.

Table 2: NTS maximum path length ($h_{\max}$ and $\overline{h_{\max}}$)

n	FI		FD	
	$h_{\max}$	$\overline{h_{\max}}$	$h_{\max}$	$\overline{h_{\max}}$
40	267	166.3	7	3.8
50	359	219.2	8	3.9
100	868	554.5	8	4.3

5. EXPERIMENTAL ANALYSIS: LRP

5.1 VNS baseline algorithm

For our comparative study, we take as a baseline algorithm a variant of the generalized VNS algorithm described in [7]. The VNS algorithm was carefully designed with LRP specific neighborhood combinations. More precisely, the baseline VNS has two standard components: random shaking and local search both using different neighborhood structures. For local search, a standard VND is used with the following neighborhoods: $\mathcal{N}_1 \cup \mathcal{N}_2$, $\mathcal{N}_3 \cup \mathcal{N}_4$, $\mathcal{N}_5 \cup \mathcal{N}_6$, $\mathcal{N}_7 \cup \mathcal{N}_8$, $\mathcal{N}_9 \cup \mathcal{N}_{10}$, and $\mathcal{N}_{11}$. For shaking, both neighborhoods $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_{11}$ are combined to produce a random neighbor each time the VND local search fails producing an improving local optimum. The maximum strength of the shaking is fixed to be a function of the problem size, namely, $n + m$. The standard VNS shaking strategy is used, i.e., if a new improving local optimum is found, shaking is rested to neighborhood 1, otherwise the next neighborhood is considered and so on until reaching the last neighborhood. This specific shaking and local search is motivated in [7] by its ability to manage the two decision levels (Location and Routing) induced by LRP.

Generally speaking, VNS is particularly interesting for our NTS study since it uses a shaking phase combined with an efficient variable neighborhood descent. Since NTS is not equipped with any specific shaking (perturbation) procedure, the goal is to study whether the backtracking component of NTS is able to efficiently escape the local optima computed in the descent phase and to effectively find better ones without any specific shaking (perturbation). Proving that *non* problem specific backtracking strategies can lead to competitive algorithms would help the design of generic search algorithms that can be applied without any problem specific tuning. In the following, starting from a randomly generated initial solution, different solution quality / computing cost trade-offs are obtained depending on NTS backtracking strategy, step function and acceptance criterion.

5.2 Solution Quality vs Computing Cost

We first examine solution quality obtained with NTS using FD and BD local step strategies compared to VNS. We select ordinal data analysis to compare the considered algorithms. For each algorithm a and each experiment ℓ, an ordinal value o_a^ℓ representing the rank of the algorithm is given. To compare the relative performance of competing algorithms, we aggregate the obtained orders for each algorithm into a unique order. We use a simple and intuitive aggregation method, known as the *Borda count* voting method. An algorithm having rank o_a^ℓ in an experiment is given o_a^ℓ points, and

the total score of an algorithm is simply the sum of its ranks over all experiments. The algorithms are then compared to their cumulative scores where the algorithm having the *smallest score* being considered as the best performing algorithm. For each instance, the ranks were computed using as a metric the solution gap to lower bound averaged over 30 trials (The lower bounds were taken from the work in [2].). In other words, for each instance, the algorithm having the i^{th} smaller average gap is ranked i and thus it is scored with i points. The final score of each algorithm is them the sum of its scores over all instances. For LRP, we consider 5 instance sets according to problem size. Each set contains 90 instances. We will consider 7 algorithms, i.e., 6 NTS variants and VNS. Thus, for a given instance set, the best (resp. worst) possible score is 90 (resp. 630), while the best (resp. worst) possible *total* score is 450 (resp. 3150). Our first results are summarized in Table 3. We can observe that for lower instances sizes, FD step strategies outperforms VNS with all backtrack strategies. However, for higher instances sizes only BU performs better than VNS where as BH is the worst performing strategy overall. We attribute this to the diversification introduced by the BU strategy. Actually, the solution quality results reported in Table 3 have a price in terms of computing cost as discussed below.

Table 3: Solution quality with Borda count voting method for LRP using NTS variants and VNS. Notation NTS-(X, Y, Z) was defined in Section 2. Acceptance criterion AA is fixed for all variants. $X \in \{BD, FD\}$ refers to step function. $Z \in \{BH, BR, BU\}$ is the backtracking strategy. In bold, we highlight the scores in favor of NTS over VNS.

(n,m)	NTS $-(*, AA, *)$						VNS
	BH		BR		BU		
	BD	FD	BD	FD	BD	FD	
$(10, 30)$	598	438	519	348	**230**	**99**	286
$(10, 20)$	607	471	490	370	**218**	**115**	248
$(5, 30)$	488	**359**	401	**266**	**141**	**91**	399
$(5, 20)$	452	**301**	**349**	**190**	**131**	**93**	403
$(5, 10)$	368	**250**	322	**225**	**203**	**115**	310
Total	2513	1819	2081	**1399**	**923**	**513**	1646

In Table 4, we report the joint solution quality, and computing cost (until termination) for NTS compared to VNS. We denote $\overline{r_{\text{eval}}}$ the ratio obtained when dividing the total number of evaluations performed by NTS by the total number of evaluations performed by VNS. We denote $n_>$ a Borda like score computed as following. For each instance, we compare the average gap to lower bound of NTS and VNS. If NTS is better we score it $+1$, in case of equality we score it 0, and otherwise -1. $n_>$ is then obtained by summing up the computed scores. Having 90 instances per problem size, the best (resp. worst) score is $+90$ (resp. -90). A positive (resp. negative, zero) score means that NTS performs better on more (resp. less, equally) number of instances. This gives a general idea on the number of instances for which NTS performs better than VNS (Taking $n_>/90$ gives the ratio of instances where NTS performs better or worst depending on $n_>$ sign). For simplicity we only report results with FD step function. Notice that Table 4 gives an idea not only about the performance of NTS compared to VNS, but also the relative performance of the different NTS strategies. One can clearly see the different trade-offs given by NTS in terms of solution quality (BH $<$ BR $\simeq$ VNS $<$ BU) and computing cost (BU $<$ VNS $\simeq$ BR $<$ BH). E.g., for lower instance sizes, BH beats VNS in both two measures. BR gives better solution quality with comparable running cost. At higher instance sizes, only BU is able to perform better than VNS in solution quality but at the price of being around 4 times slower.

Table 4: Solution quality and computing cost of NTS with FD and AA strategies compared to VNS.

(n,m)	NTS $-(FD, AA, *)$					
	BH		BR		BU	
	$n_>$	$\overline{r_{\text{eval}}}$	$n_>$	$\overline{r_{\text{eval}}}$	$n_>$	$\overline{r_{\text{eval}}}$
$(10, 30)$	-60	0.50	-37	0.84	73	5.00
$(10, 20)$	-74	0.45	-52	0.77	56	3.75
$(5, 30)$	20	0.45	44	0.78	70	4.19
$(5, 20)$	40	0.76	53	1.32	59	6.18
$(5, 10)$	19	0.82	24	1.40	47	4.67
Total	-55	0.60	32	1.02	305	4.76

5.3 Speeding up the search

In this section, we give the results we have obtained by running NTS with acceptance criterion AT. Recall that with the AT strategy a neighborhood branch is explored depending on the best locally observed fitness f_s^{best}, and with a probability which is inversely proportional to the trajectory path length. Results with FD step function are summarized in Table 5. One can clearly see that the AT strategy has the effect of speeding-up the search (compared to results with AA given in Table 4). This is rather expected since neighborhood branches producing solutions with poor quality compared to other neighborhoods are likely to be pruned as we get deeper in the search path. While strategy AT allows us to speed up the search, it has two 'side-effects'. Firstly, compared to AA, AT produces less high solution quality for all backtracking strategies. Secondly and for the largest instances, AT is no more competitive against the finely tuned VNS even using the most effective BU backtracking strategy.

Table 5: Solution quality and computing cost of NTS with FD and AT compared to VNS.

(n,m)	NTS $-(FD, AT, *)$					
	BH		BR		BU	
	$n_>$	$\overline{r_{\text{eval}}}$	$n_>$	$\overline{r_{\text{eval}}}$	$n_>$	$\overline{r_{\text{eval}}}$
$(10, 30)$	-80	0.22	-66	0.36	-9	1.03
$(10, 20)$	-88	0.23	-82	0.35	-38	0.88
$(5, 30)$	-22	0.22	13	0.35	66	0.92
$(5, 20)$	16	0.40	42	0.63	55	1.54
$(5, 10)$	0	0.53	14	0.81	32	1.53
Total	-174	0.32	-79	0.50	106	1.18

5.4 Time to best with step function FI

In accordance with the results obtained for SMTWTP, we found that NTS combined with FI step function is able to give very good results for LRP. In the following, we report only our findings when running NTS for a maximum number of evaluations, namely, 10^7 evaluations. For all competing algorithms, VNS included, we study the number of evaluations it takes for an algorithm to find the best fitness solution. Using acceptance conditions AA and AT, we report the values of $n_>$ and $\overline{r'_{\text{eval}}}$ which is now the ratio of the number of fitness evaluations it takes for NTS and VNS to find the best

Table 6: Solution quality and computing cost of NTS with FI and AA compared to VNS.

(n,m)	NTS $-(FI, AA, *)$					
	BH		BR		BU	
	$n_>$	$\overline{r'_{\text{eval}}}$	$n_>$	$\overline{r'_{\text{eval}}}$	$n_>$	$\overline{r'_{\text{eval}}}$
$(10, 30)$	-25	0.67	-31	0.69	-24	0.70
$(10, 20)$	44	1.87	58	2.14	67	2.48
$(5, 30)$	70	0.26	70	0.26	70	0.26
$(5, 20)$	59	0.40	59	0.44	59	0.42
$(5, 10)$	46	0.18	48	0.24	50	0.27
Total	194	0.68	204	0.75	222	0.83

Table 7: Solution quality and computing cost of NTS with FI and AT compared to VNS.

(n,m)	NTS $-(FI, AT, *)$					
	BH		BR		BU	
	$n_>$	$\overline{r'_{\text{eval}}}$	$n_>$	$\overline{r'_{\text{eval}}}$	$n_>$	$\overline{r_{\text{eval}}}$
$(10, 30)$	11	1.49	35	2.64	-27	0.69
$(10, 20)$	8	0.84	33	1.45	69	2.47
$(5, 30)$	70	0.39	70	0.48	70	0.26
$(5, 20)$	59	0.34	59	0.40	59	0.44
$(5, 10)$	42	0.16	48	0.19	50	0.24
Total	190	0.64	245	1.03	221	0.82

solution. Our results are summarized in Tables 6 and 7. In accordance with the results obtained with FD, different trade-offs are obtained. For instance, the computing cost of BH is better than BR which is better than BU. For all instance sets, but for size $(10, 30)$, BU beats all other strategies in terms of solution quality. Actually, for instance set $(10, 30)$ strategy BU needs more time to find high quality solutions, i.e., BU is an exploration oriented strategy which needs more time to converge but produces very high solution quality. Moreover, we can state that overall instance set NTS with FI strategy performs better than VNS. In particular, backtracking strategies BH and BR provides very competitive results both in computing cost and solution quality especially when combined with the adaptive acceptance criterion AT.

6. CONCLUSION

In this paper, by operating at the level of the tree induced by a set of several different neighborhood structures, we introduced a backtracking traversal algorithm called NTS and studied some of its variants. Compared to standard VND where neighborhood ordering can be critical, NTS is able to find its way by simply piping different neighborhoods dynamically at runtime. Compared to VNS where shaking is crucial, backtracking in NTS is able to escape local optima searching for promising neighborhood paths. However, since exploring the neighborhood tree in an exhaustive manner could be intractable, NTS components (step function, neighborhood selection, branching, accepting, backtracking) have to be carefully combined in order to obtain a good compromise between solution quality and computing cost. In particular, the NTS variants described in this paper are based on the following two intuitive claims: (i) the more we are deep in the neighborhood tree, the more it is likely to find better local optima (intensification) (ii) the less we are deep in the neighborhood tree, the more it is likely to explore new search regions and thus to go forward through new high quality solutions (diversification). Generally speaking, we claim that new adaptive backtracking strategies combined with new adaptive acceptance criteria would be the key ingredients providing the efficient balance between intensification and diversification in NTS. We believe that this is a challenging and interesting open question which deserves further investigations.

7. REFERENCES

[1] http://people.brunel.ac.uk/ mastjjb/jeb/orlib/wtinfo.html.

[2] M. Albareda-Sambola, J. A. Diaz, and E. Fernandez, *A compact model and tight bounds for a combined location-routing problem*, Computers & Operations Research **32** (2005), no. 3, 407 – 428.

[3] M. D. Besten, T. Stutzle, and M. Dorigo, *Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem*, EvoWorkshops, 2001, pp. 441–452.

[4] E. K. Burker, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, *A classification of hyper-heuristic approaches*, Handbook of Meta., 2010, pp. 449–468.

[5] K. Chakhlevitch and P. I. Cowling, *Hyperheuristics: Recent developments*, Adaptive and Multilevel Metaheuristics, 2008, pp. 3–29.

[6] P. I. Cowling, G. Kendall, and E. Soubeiga, *A hyperheuristic approach to scheduling a sales summit*, 3th Int. Conf. on Pract. and Th. of Auto. Timetabling, 2001, pp. 176–190.

[7] H. Derbel, B. Jarboui, H. Chabchoub, S. Hanafi, and N. Mladenovic, *A variable neighborhood search for the capacitated location-routing problem*, 4th IEEE Int. Conf. on Logistics (LOGISTIQUA),, 2011, pp. 514 –519.

[8] H. Derbel, B. Jarboui, S. Hanafi, and H. Chabchoub, *Genetic algorithm with iterated local search for solving a location-routing problem*, Expert Syst. Appl. **39** (2012), no. 3, 2865–2871.

[9] C. Duhamel, Lacom P., C. Prins, and C. Prodhon, *A grasp&els approach for the capacitated location-routing problem*, Comput. Oper. Res. **37** (2010), 1912–1923.

[10] L. Gaspero and A. Schaerf, *Neighborhood portfolio approach for local search applied to timetabling problems*, J. of Mathematical Modelling and Algorithms **5** (2006), 65–89.

[11] Martin Josef Geiger, *On heuristic search for the single machine total weighted tardiness problem - some theoretical insights and their empirical verification*, EJOR **207** (2010), no. 3, 1235–1243.

[12] A. Goeffon, J.-M. Richer, and J.-K. Hao, *Progressive tree neighborhood applied to the maximum parsimony problem*, IEEE/ACM Trans. Comput. Biol. Bio. **5** (2008), 136–145.

[13] A. Grosso, F. Della Croce, and R. Tadei, *An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem*, Operations Research Letters **32** (2004), no. 1, 68 – 72.

[14] P. Hansen, N. Mladenović, and P. J. Moreno, *Variable neighbourhood search: methods and applications*, Annals of Operations Research **175** (2010), 367–407.

[15] H. H. Hoos and T. Stützle, *Evaluating las vegas algorithms: pitfalls and remedies*, 14th Conf. on Uncertainty in artificial intelligence, 1998, pp. 238–245.

[16] B. Hu and G. Raidl, *Variable neighborhood descent with self-adaptive neighborhood-ordering*, 7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Meta., 2006.

[17] Z. Lü, F. Glover, and J.-K. Hao, *Neighborhood combination for unconstrained binary quadratic problems*, MIC Post-Conference Book, 2011, pp. 49–61.

[18] Z. Lü, J.-K. Hao, and F. Glover, *Neighborhood analysis: a case study on curriculum-based course timetabling*, J. of Heuristics **17** (2011), 97–118.

[19] H. Min, V. Jayaraman, and R. Srivastava, *Combined location-routing problems: A synthesis and future research directions*, EJOR **108** (1998), no. 1, 1 – 15.

[20] G. Nagy and S. Salhi, *Location-routing: Issues, models and methods*, EJOR **177** (2007), no. 2, 649–672.

[21] E. Özcan, B. Bilgin, and E. E. Korkmaz, *A comprehensive analysis of hyper-heuristics*, Intell. Data Anal. **12** (2008), 3–23.

[22] J. Puchinger and G. Raidl, *Bringing order into the neighborhoods: relaxation guided variable neighborhood search*, J. of Heuristics **14** (2008), 457–472.